下载APP

21 | 压测平台：高效搭建 GoReplay 压测平台

2021-12-04 高楼

《全链路压测实战30讲》 课程介绍 >

讲述：高楼

时长 17:46 大小 16.27M

你好，我是高楼。

从这里开始，我们就进入全链路压测的第三部分“实践环境”了。

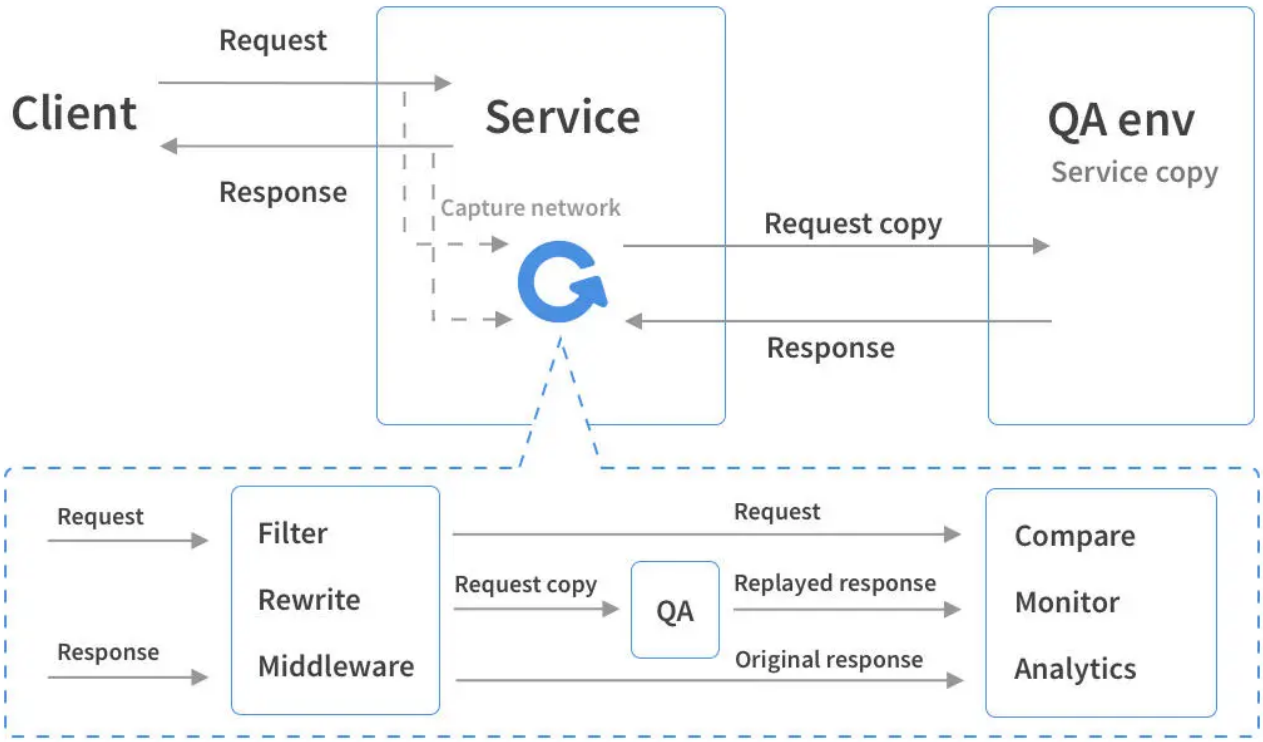
在接下来的三讲，我会详细讲讲如何搭建压测平台。要搭建压测平台，首先，我们就要选择一款适合自己项目的流量工具。在第 6 讲，我们已经详细了解了流量工具的选型。综合对比评估后，最后我们这个项目选择了老牌流量回放工具 GoReplay，因为它简单、轻量、热度够，而且完全能满足我们目前项目的要求。

所以这节课，我们就来聊一聊怎样搭建 GoReplay 压测平台。如果你对 GoReplay 非常熟悉，那可以跳过下面的概念讲解，如果你之前没有接触过 GoReplay，可以跟着我一起先大概了解下这个工具。

GoReplay 是什么？

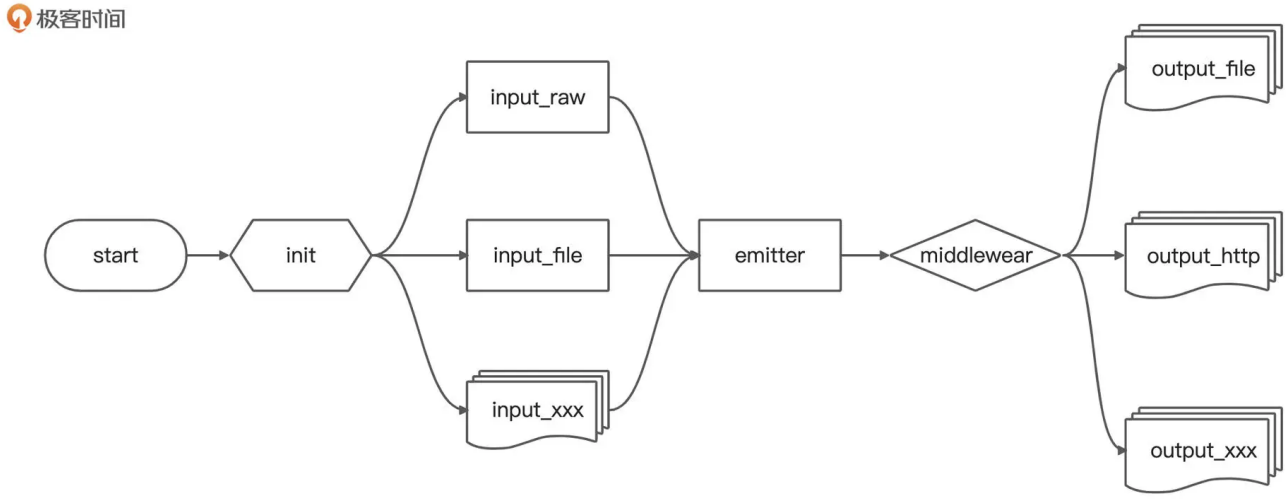
GoReplay 的简称是 Gor，它是一个简单的 TCP/HTTP 流量录制和回放的工具，主要用 Golang 语言编写。

GoReplay 的工作原理图可以参考下面这张：



GoReplay 可以在服务器上启动一个 Gor 进程，它负责的工作包括**监听、过滤、转发、回放**等。通过监听网卡，它可以直接录制请求。后续它还支持实现**流量回放、压力测试、性能监控**等功能。

如果我们简化一下核心流程图，它会是下面这个样子：



GoReplay 从数据流中抽象出了两个概念，即**输入（Input）和输出（Output ）**，它用这两个概念来表示数据来源与去向，统称为 Plugin。

GoReplay 用介于输入和输出模块之间的 Middleware（中间件）实现拓展机制，而 Emitter 则是核心处理器，实现对于 Input 输入流的读取，并判断是否需要进行 Middleware 的处理、请求修改等。完成这一步后，Emitter 会异步复制流量到所有 Output，同时将所有 Output 中有 Response 的数据复制到所有 Outputs 中。

Golang 环境安装

好了，理清 GoReplay 的工作原理之后，我们来看下 Golang 的环境安装。只有安装好 Golang 才能运行 GoReplay。

第一步，安装 Golang 及相关依赖环境，你可以在官网或者 Go 的一些中文网站上下载安装包。

我这里下载的是: go1.15.5.linux-amd64.tar.gz。

第二步，解压到服务器 /usr/local 目录下。

复制代码

```
rm -rf /usr/local/go && tar -C /usr/local -xzf go1.15.5.linux-amd64.tar.gz
```

第三步，配置环境变量。

复制代码

```
# 打开
vim /etc/profile

# 添加
export GOROOT=/usr/local/go
export PATH=$PATH:$GOROOT/bin

# 编译生效
source /etc/profile
```

最后，我们验证 Golang 环境是否生效。

复制代码

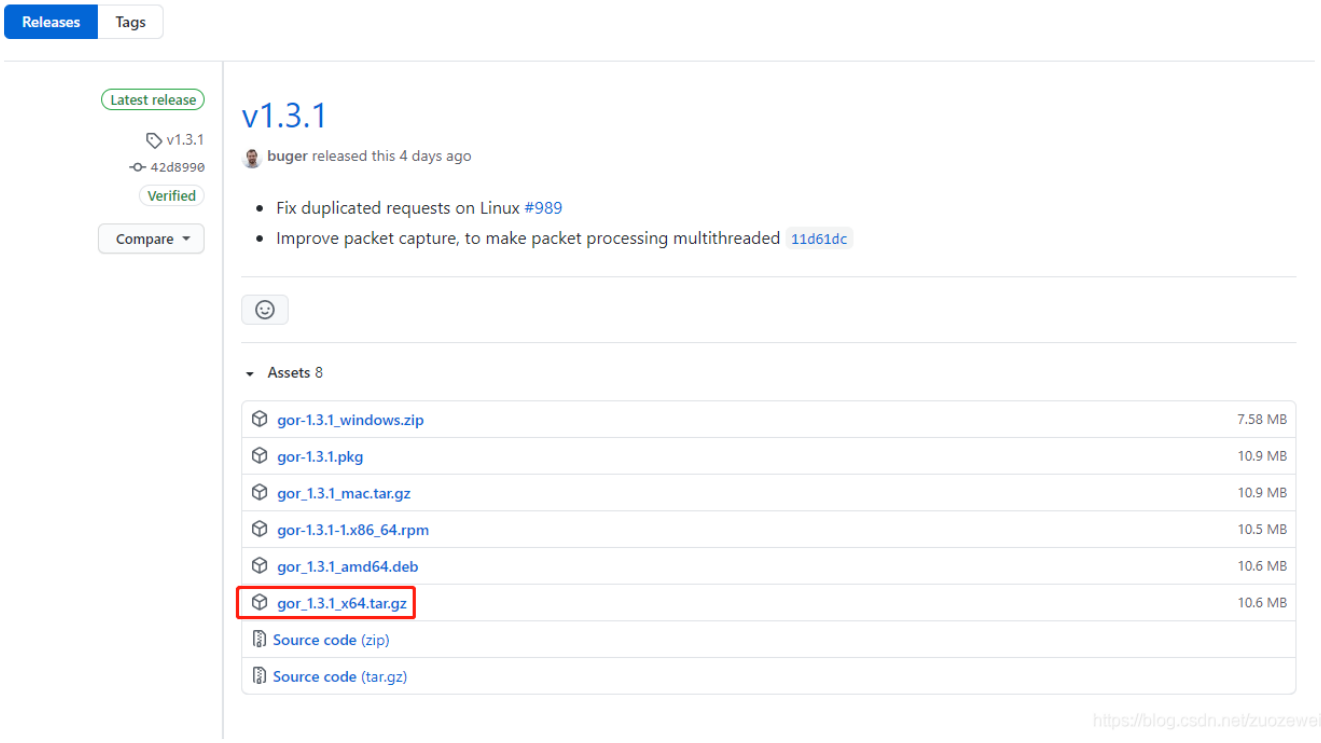
```
[root@vm ~]# go version
go version go1.15.5 linux/amd64
```

我们看到，版本已经显示成功了，这说明我们的环境已经安装成功了。

GoReplay 安装

接下来，我们安装 GoReplay。在我们这个项目中，因为要在网关服务上录制流量，所以需要在网关容器内安装 GoReplay。

我们从 https://github.com/buger/gor/releases 下载 Gor 二进制文件（提供 Windows、Linux x64 和 macOS 的预编译二进制文件），也可以自己源码编译安装。



这里，我简单演示一下如何通过 curl 下载二进制包。

```
$ curl -L -O https://github.com/buger/goreplay/releases/download/v1.3.1/gor_1.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
                                 Dload  Upload   Total   Spent    Left  Speed
100   626  100   626    0     0    741      0 --:--:-- --:--:-- --:--:--   741
100 10.5M  100 10.5M    0     0  3258k      0  0:00:03  0:00:03 --:--:-- 5951k

$ tar xvzf gor_1.3.1_x64.tar.gz
gor
```

解压缩包后，我们可以直接从当前目录运行 Gor 二进制文件，或者更方便的是将二进制文件复制到系统的 PATH（对于 Linux 和 macOS，它可以是 /usr/local/bin）中。

同时我们也要记得安装抓包工具 libpcap：

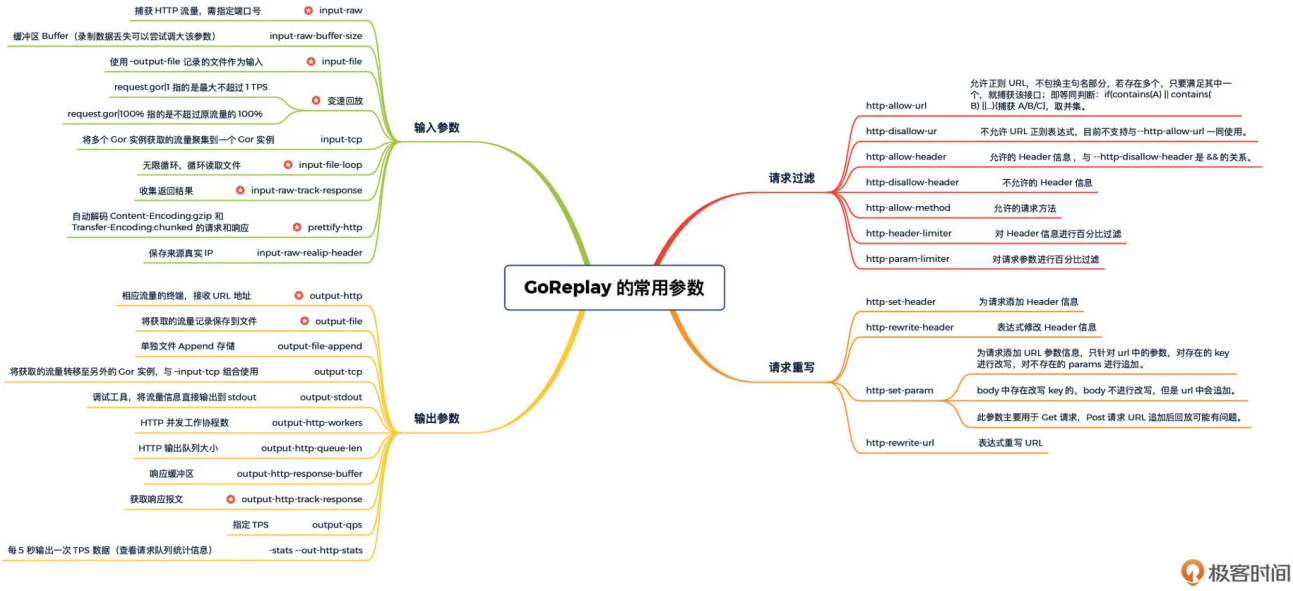
```
yum install libpcap libpcap-devel
```

这里我演示的是 yum 安装方式。

这样，我们就可以使用 Gor 命令进行各种操作了。

在正式介绍流量录制和流量回放的逻辑之前，我还想带你熟悉一下 GoReplay 常用的参数。市面上对这些常用参数的介绍，比较齐全的不多，所以我在这里做个总结。

这里我按照使用类型划分，整理了常用的参数：



其中标星的参数是压测中的必选项。

好了，知道了常用的参数，特别是在压测中必然会用到的参数，接下来我们就看下如何在具体项目中使用。

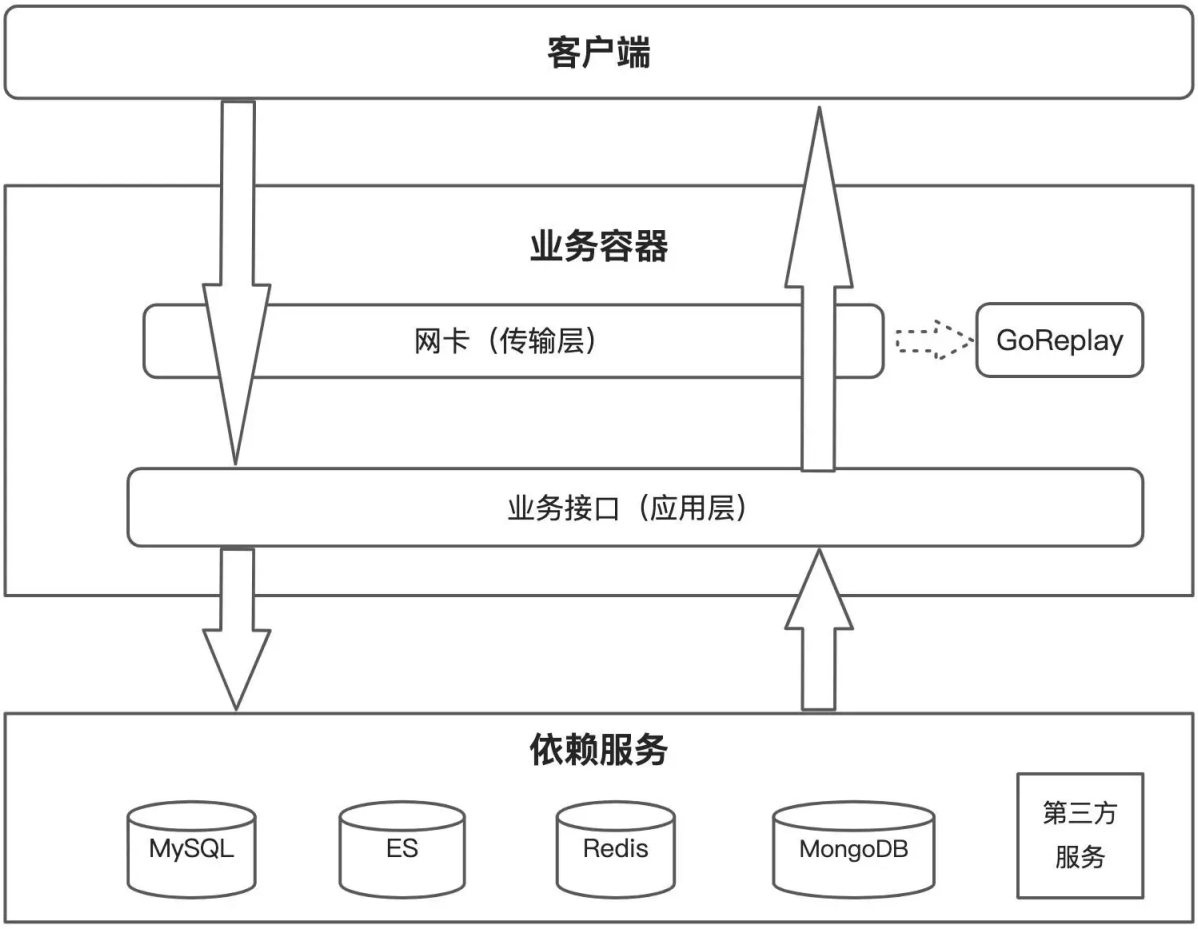
流量录制

我们知道流量录制工具的核心原理就是将线上集群环境的流量复制多份发送到指定的仓库，然后使用指定压力机器进行放大回放以达到压力测试的目的。

那么 GoReplay 是如何实现流量复制的呢？

核心原理

你可以先来看下这张 GoReplay 的录制原理图：



GoReplay 通过调用 Google/gopacket 来实现抓包，这里的 gopacket 是基于 libpcap 库的。GoReplay 可以做到捕捉指定端口的网卡流量，它既可以实现 TCP 协议（RAW_SOCKET）的抓包，也可以实现 HTTP 的录制、回放，同时还支持多实例之间的级联。

录制的时候，每个 TCP 包被抽象为 packet。当数据量较大、数据需要被分拆成多个包发送时，接收端需要把这些包按顺序组合起来，同时还要处理乱序、重复等问题，保证向下传递的是一个完整无误的数据包。这些逻辑统封装在了 tcp_message 中，tcp_message 与 packet 是一对多的关系。后续会将 tcp_message 中的数据取出，打上标记，传递给中间件（如果有）或者是 Output 插件。

要做到这些，我们来看下 GoReplay 基于 libpcap 库的核心函数。

首先，GoReplay 的 capture.go 类中定义了 pcap 处理引擎函数：

复制代码

```
func (l *Listener) activatePcap() error {
  var e error
  var msg string
  for _, ifi := range l.Interfaces {
    var handle *pcap.Handle
    handle, e = l.PcapHandle(ifi)
    if e != nil {
      msg += ("\n" + e.Error())
      continue
    }
    l.Handles[ifi.Name] = packetHandle{
      handler: handle,
      ips:     interfaceIPs(ifi),
    }
  }
  if len(l.Handles) == 0 {
    return fmt.Errorf("pcap handles error:%s", msg)
  }
  return nil
}
```

capture.go 类描述的几种引擎包括 libpcap、pcap_file、raw_socket 等：

复制代码

```
// Set is here so that EngineType can implement flag.Var
func (eng *EngineType) Set(v string) error {
  switch v {
  case "", "libpcap":
    *eng = EnginePcap
  case "pcap_file":
    *eng = EnginePcapFile
  case "raw_socket":
    *eng = EngineRawSocket
  case "af_packet":
    *eng = EngineAFPacket
  default:
    return fmt.Errorf("invalid engine %s", v)
  }
  return nil
}
```

好了，知道了录制的原理，接下来我们看下如何在具体项目中做录制。

录制实例

这里我们演示的例子是除健康检查（/actuator/health）之外，录制网关服务的所有接口请求。

录制逻辑图：

我们主要会使用下面这些命令：

复制代码

```
# --input-raw：捕获指定端口 HTTP 流量
# --output-file：将获取的流量记录保存到文件
# -output-file-append：单独文件 Append 存储
# --http-set-header：为请求添加 header 信息
# --input-raw-track-response：收集返回结果
# --prettify-http：自动解码 Content-Encoding:gzip 和 Transfer-Encoding:chunked 的
# --http-disallow-url：不允许正则 URL
sudo nohup ./gor --input-raw :8201 \
  --output-file=request-mall-all.gor \
  -output-file-append \
  --http-set-header "dunshan:7DGroup" \
  --http-set-header "User-Agent:Replayed-by-Gor" \
  --input-raw-track-response \
  --prettify-http \
  --http-disallow-url /actuator/health > /dev/null 2>&1 &
```

这些命令的意思是我们要监听网关实例的 8201 端口，录制除 /actuator/health 之外的所有请求，然后捕获响应报文并把请求追加到唯一文件中，像这里生成的流量文件名叫“request-mall-all.gor”。同时，我们还要对 Header 信息重写并打上压测标记“dunshan:7DGroup”。

这里有一些需要说明的是：

sudo：以 root 身份运行，这是监听网络流量所必需的（除非另行配置）；

nohup：允许该命令后台运行；

/dev/null：让 GoReplay 进程监听 /dev/null。/dev/null 表示的是一个黑洞，用于丢弃不需要的输出。如果日志文件太大，并且你对自己的设置有足够的信心，可以选择重定向到 /dev/null。

2>&1：将 stderr（标准错误）输出重定向到 stdout（标准输出），允许我们在上面指定的同一日志文件中收集错误输出。

&：在后台运行进程。这样我们就可以在运行 GoReplay 时继续使用 Shell 了。

命令执行后，输出的是下面的结果：

复制代码

```
[root@s12 ~]# sudo nohup ./gor --input-raw :8201 \
  --output-file=request-mall-all.gor \
  -output-file-append \
  --http-set-header "dunshan:7DGroup" \
  --http-set-header "User-Agent:Replayed-by-Gor" \
  --input-raw-track-response \
  --prettify-http \
  --http-disallow-url /actuator/health > /dev/null 2>&1 &
[1] 26251
```

这里显示的数字 26251 是 Gor 程序的进程 PID，在我们录制完成后，可以使用这个 PID 终止 Gor 的录制进程。

现在， Gor 已经开始流量录制了，这段时间网关转发的所有请求会被录制。

在录制了一段时间的流量后，我们可以执行下面的命令终止 Gor 的录制。录制终止之后，可以看到执行录制指令的目录下有了一份文件名为 “request-mall-all.gor ” 的流量文件。

复制代码

```
# 手动输入 PID 方式
sudo kill -9 ${gor 进程 PID}

# 自动化获取 PID 方式
for i in `ps -ef|grep gor|grep -v grep |awk '{print $2}'`; do  kill -9 $i; don
```

其实如果我们考虑限时录制的话，也可以参考下面这段命令：

复制代码

```
# timeout 60 表示只录制 60 秒后自动停止
sudo nohup timeout 60 ./gor --input-raw :8201 \
  --output-file=request-mall-all.gor \
  -output-file-append \
  --http-set-header "dunshan:7DGroup" \
  --http-set-header "User-Agent:Replayed-by-Gor" \
  --input-raw-track-response \
  --prettify-http \
  --http-disallow-url /actuator/health > /dev/null 2>&1 &

# 按小时切割日志文件，并且开启日志追加模式，不会进行日志默认小分片
sudo nohup timeout 60 ./gor --input-raw :8201 \
  --output-file=request-mall-all-%Y-%m-%d-%H.gor \
  -output-file-append \
  --http-set-header "dunshan:7DGroup" \
  --http-set-header "User-Agent:Replayed-by-Gor" \
  --input-raw-track-response \
  --prettify-http \
  --http-disallow-url /actuator/health > /dev/null 2>&1 &

# 按小时切割日志文件，并且开启日志追加模式，不会进行日志默认小分片，.gz 压缩文件格式
sudo nohup timeout 60 ./gor --input-raw :8201 \
  --output-file=request-mall-all-%Y-%m-%d-%H.gz \
  -output-file-append \
  --http-set-header "dunshan:7DGroup" \
  --http-set-header "User-Agent:Replayed-by-Gor" \
  --input-raw-track-response \
  --prettify-http \
  --http-disallow-url /actuator/health > /dev/null 2>&1 &
```

接下来，我们尝试打开“request-mall-all.gor ”的流量文件，分析一下里面的结构。

因为 GoReplay 用三个猴头 🙊🙉🙈 作为请求分隔符，所以我们的流量文件是下面这个样子：

复制代码

```
# 请求类型，请求 ID 为 80a820090a6423c03994b9f3，时间戳
1 80a820090a6423c03994b9f3 1635694104268073719 0
# 请求的类型及 URL 地址
POST /api/member/sso/login HTTP/1.1
Connection: keep-alive
# 压测标记
dunshan: 7DGroup
Content-Length: 34
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

```
Host: 172.31.184.225:30001
# header 重写
User-Agent: Replayed-by-Gor

# 请求参数
username=7dTest006&password=123456

🐵🙈🐵 # 请求分隔符

# 响应类型，请求 ID 为 80a820090a6423c03994b9f3，时间戳
2 80a820090a6423c03994b9f3 1635694104291403073 392247
HTTP/1.1 200 OK
Content-Length: 1053
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Date: Sun, 31 Oct 2021 15:28:24 GMT
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1 ; mode=block
Referrer-Policy: no-referrer

# 响应体
{"code":200,"message":"操作成功","data":{"token":"eyJhbGciOiJSUzI1NiIsInR5cCI6I
```

从流量文件中可以看出，对于每个请求，应该收到 3 个有效 Payload(1-Request, 2-Response, 3-Replayed Response)，它们具有相同的请求 ID（Request 和 Response 具有相同的)。

到此，流量录制就结束了。

还有一点需要注意的是，在录制大流量的的时候，有可能会出现丢失部分请求的情况。这是因为 Gor 是基于 pcap 和操作系统 BufferSize 进行录制的，当 BufferSize 溢出的时候，请求就会丢失。

这个时候我们必须设置参数：–input-raw-buffer-size。

复制代码

```
  -input-raw-buffer-size value
        Controls size of the OS buffer which holds packets until they dispatch
```

我们可以在 capture.go 中找到 BufferSize 相关的这段代码：

复制代码

```
if l.BufferSize > 0 {
  err = inactive.SetBufferSize(int(l.BufferSize))
  if err != nil {
    return nil, fmt.Errorf("handle buffer size error: %q, interface: %q", er
  }
}
if l.BufferTimeout == 0 {
  l.BufferTimeout = 2000 * time.Millisecond
}
err = inactive.SetTimeout(l.BufferTimeout)
if err != nil {
  return nil, fmt.Errorf("handle buffer timeout error: %q, interface: %q", e
}
handle, err = inactive.Activate()
```

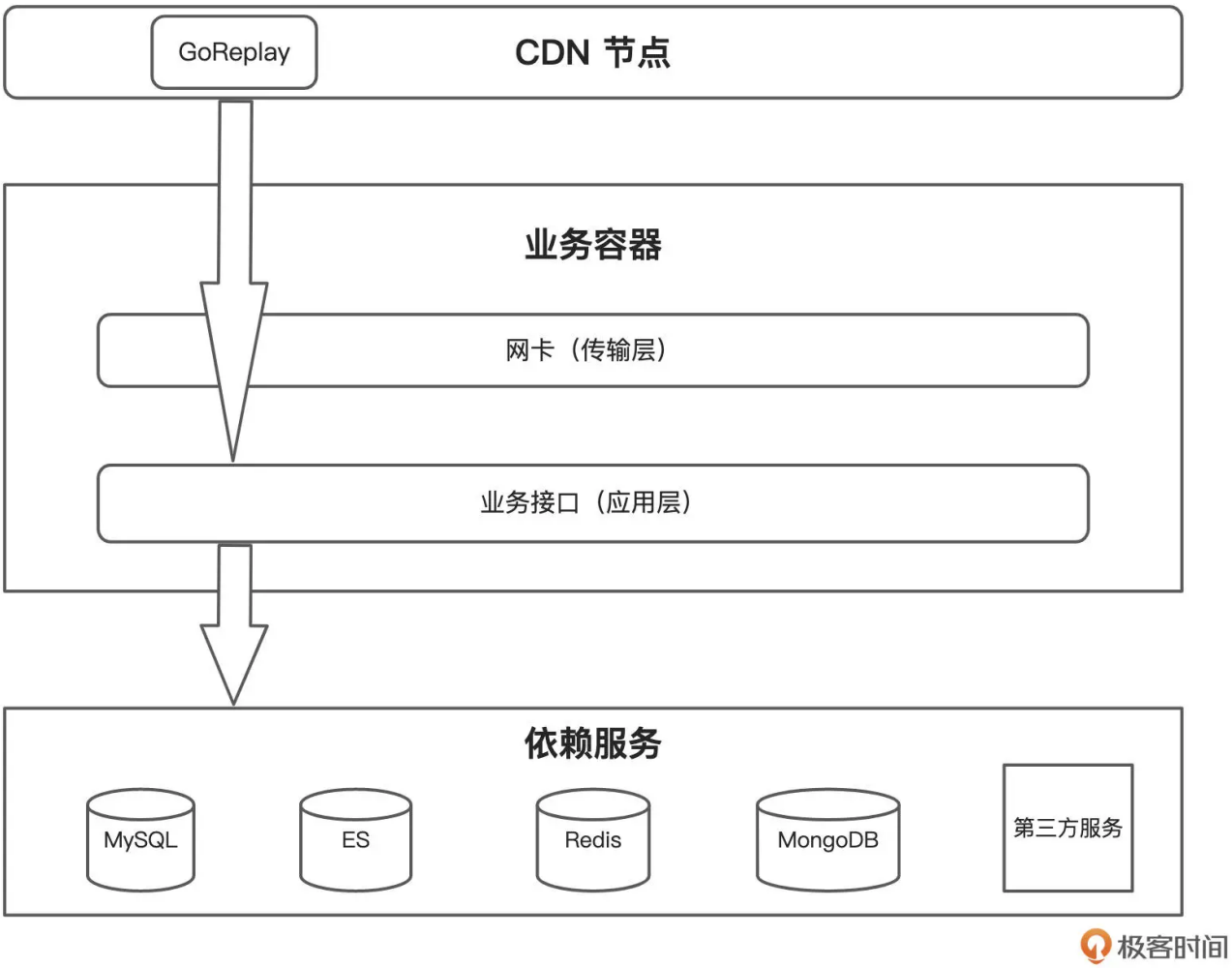
要注意的是，操作系统缓冲区（ BufferSize ）的默认值是用来保存数据包的，它在不同的操作系统上有所不同。例如，在 Linux 上这个值是 2 MB，在 Windows 是 1 MB。如果设置更大的缓冲区或减少 MTU（接口层的最大数据包数据大小）不能减少丢失的请求数据，那么很有可能是 GoReplay Bug。

流量回放

接下来，我们再看下流量回放。流量回放的核心原理是通过 Limiter 实现变速回放，通过 Output 插件输出到目标机器。

这里我们主要使用的是 HTTP 输出的插件 output_http.go ，它通过实现 HTTP 协议，进而实现 io.Writer 接口，最后根据配置注册到 Plugin.outputs 队列里。

GoReplay 回放原理图：



流量回放主要通过 Limiter 类执行各种变速。默认情况下，Gor 会创建一个动态协程池。初始线程数是 10 ，因为线程数量（ N ）等于该线程池的队列长度，所以当 HTTP 输出队列长度大于 10 时，Gor 会创建更多的协程数。

在产生 N 个协程的请求得到满足之前，不会再有协程创建。如果动态协程数当时不能处理消息，它将会睡眠一会。如果动态协程数无法处理消息，它就会死亡。我们可以用 --output-http-workers=20 选项指定并发协程数。

这里我们演示的例子是，倍数回放网关服务的所有接口请求。

回放逻辑图：

我们主要会使用下面这些命令：

```
# --input-file：从文件中获取请求数据，重放的时候 100x 倍速
# --input-file-loop：无限循环，而不是读完这个文件就停止
# --output-http：发送请求到网关
# --output-http-workers：并发 100 协程发送请求
# --stats --output-http-stats：每 5 秒输出一次 TPS 数据
# --output-http-track-response：获取响应报文
$ sudo ./gor --input-file 'request-mall-all.gor|10000%' \
--input-file-loop \
--output-http 'http://10.96.136.36:8201' \
--output-http-workers 100 \
--stats \
--output-http-stats \
--output-http-track-response
```

因为我们是性能压测，可以要求无限循环。可以看到，我们向网关服务回放了 100 倍的流量。

总结

好了，这节课就讲到这里。刚才，我们一起梳理了 GoReplay 的基本概念、核心原理、常用参数。我们还一起认识了 GoReplay 两个主要的功能：流量录制和流量回放。下面几个知识点希望你能记住。

1. GoReplay 主要依赖 Golang 环境运行，它通过命令行组织各个参数进行具体的操作；
2. GoReplay 底层依赖的是 libpcap 库， Gor 进程可以负责所有的工作，包括监听、过滤、重写、回放等；
3. 通过参数：-stats 和 --out-http-stats，GoReplay 原生只支持由控制台统计发送请求队列信息，如果想要更复杂的测试统计还需要我们自己去埋点丰富监控指标；
4. 使用 Middleware 可以让我们在输入输出间做一些复杂的请求增强工作。

下一节课，我们将进入具体实践环节，我会通过案例演示如何做动态数据关联改造工作。

思考题

学完这节课，请你思考两个问题：

1. 你有没有使用过 GoReplay，谈谈你使用这个工具的一些心得。
2. 相比 Java 语言的压测工具，你觉得 Golang 的压测工具有什么优势？

欢迎你在留言区与我交流讨论。当然了，你也可以把这节课分享给你身边的朋友，他们的一些想法或许会让你有更大的收获。我们下节课见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 20 | Mock：如何屏蔽第三方接口的影响？

更多学习推荐

精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。